# Avoiding the Technology Acquisition Minefield

**Alfred J. Barchi**
ajb@ajbinc.net
http://www.ajbinc.net/

We've all heard stories about government waste, such as $50 screwdrivers and $600 toilet seats. But, have you ever asked how much, if any, of that $600 unit cost for a toilet seat was actually profit to the contractor and how much went to pay for government red tape? Did our government really write a contract to pay $600 apiece for toilet seats, or did the contractor run into serious technical problems meeting the mil-spec requirements (e.g., resistance to jungle rot, stability at extreme temperatures, etc.) and the government bailed them out?

In case you're wondering what this has to do with investing in technology, the fact is that even the simplest software-based system is far more complex than a toilet seat and takes far longer to develop. Unforeseen technical problems can and do sap many development efforts of any potential for profitability. And the worst part is that the way many development programs are structured, you don't even see it coming until well into the development cycle.

Development efforts have three components that are tightly intertwined: engineering management, configuration management and engineering. Engineering management includes all matters that involve managing cost and schedule, including such things as personnel management, program management, sub-contractor management, facilities management, etc. Configuration management, a.k.a. baseline management, a.k.a. "engineering process," deals with defining what the product is, what is supposed to accomplish, how it works, tracking changes to the product, etc. And of course, engineering deals with the actual design, development and maintenance of the product.

If any of these three areas is deficient, you will have serious problems.

**Engineering Management** I was once called in to assess an organization that was developing a classified program for the government. By the time I arrived, they had already shipped their product. It had several thousand known problems; a stack of problem reports several feet high that hadn't been entered into the system yet, and over 150 negative QA reports outstanding. They were in serious trouble.

The problem was that they had started to get behind schedule and their management panicked and sacrificed their configuration management. For example, their engineering review board was rubberstamping changes to the baseline weeks after they had been implemented. They eventually reached the point where nobody in their organization knew what the baseline was anymore. Compounding the problem was that the QA group reported to the project manager, rather than the general manager. And the project manager, who was graded on cost and schedule performance, simply ignored them. He was intent on minimizing immediate costs, rather than *lifecycle costs*.

You might think that this program's problems were really due to poor configuration management practices. If so, then in this case you would be wrong. It's true that the program had tremendous configuration management problems, but they were the result of a lack of discipline and poor structure on the part of the engineering management. If you had invested in this program, you would have lost your shirt – the government certainly did.

Another time, we were having problems with a sub-contractor missing its milestones. I was sent, along with a team of people, to investigate. When I got to the sub-contractor's facility, my first question to them was, "Have you won any large contracts lately?" It turned out that they had, and all of the people who were supposed to be working on our contract had been transferred to the new contract. We further learned that they had run into technical difficulties on our job and were now out of money for it, but they were afraid to own up to the fact. Fortunately, we caught this situation early enough to take corrective action and minimize the damage. The program eventually turned out well.

**Configuration Management** I once worked for a company where we had weekly meetings every Tuesday to review our contracts and try to find ways to satisfy our customers' needs sufficiently to avoid being sued by them. The problem was that our sales force didn't understand our product and was selling things that didn't exist. *"Documentation!? We don't need no steenking documentation!"* Our management was simply unwilling to pay the cost of documenting and maintaining our baselines. I don't know what became of this company. I quit, along with most of their engineering staff. If you had invested heavily in this company, I guarantee that you wouldn't get much sleep at night.

I was called in one time to assess a troubled development program that supposedly had "configuration management" problems. What I found was that their configuration management group was doing everything correctly, and that they were capable of managing as many configurations as the engineering group could throw at them. The problem was that there were simply too many configurations, and the engineering group lacked the infrastructure necessary to deal with all of them. Subsequently, they took my advice and cut the number of configurations by 75%, and got back on track.

I've also had dealings with a company that worshipped "engineering process" and worked constantly to improve it. Their concept was to make their process so robust that they could save money by reducing the quality and cost of their workforce. Of course, it doesn't work this way. Their people were applying the engineering process by rote, without any real understanding. They would crank out reams of documentation for the sake of implementing the process. But their designs tended to consist of nothing but a regurgitation of the functional requirements specifications. They always got into trouble right after a project's "Critical Design Review," when they would start trying to implement their design. Their developers would have to work to functional requirements, without any clear understanding of how their piece of the system interacted with other parts of the system being developed by other members of their own team or other teams.

The sad part is that this company had a lot of very talented engineers doing a lot of high quality engineering work to produce these lousy baselines. They produced major cost and schedule overruns on almost every program that I am aware of them developing. Of course, their primary customer was the U.S. government, and they were constantly getting bailed out. When they tried to branch out into the commercial arena, applying the same process, they failed miserably.

**Engineering** The single most significant failure I see in engineering is the tendency of developers to minimize immediate cost at the expense of lifecycle cost. The corners we cut today frequently come back to haunt us. I was once involved in the development of a middleware product that cost millions of dollars to produce over a period of several years. I wrote their initial requirements specification. In it I included the capability to route a message to a backup destination in case the message couldn't be delivered to the primary destination. I also wrote the initial specifications for another middleware component of the system. This was a modular expert system-like component responsible for controlling processes and applications distributed across multiple platforms. The concept was to handle communications failures in the middleware, minimizing the involvement of the applications that ran on top of it. If a message couldn't be delivered to its primary destination, it could then be sent to an alternate location, including the process control component, for disposition. You could route the message to a backup location, you could issue an alarm to an operator, you could send the message to a modular mailbox process for later delivery, you could start a process and then deliver the message to it or you could change the state of the application, the site or the system, all based on how you wrote the scripts in the process control module. None of this would require changes to the message routing system or the applications using it. Better still, none of these capabilities impacted the normal timeline, because they only got invoked when there was a problem.

I then went on to bigger and better things. When I next looked in on this program, they had deleted the backup routing requirement along with all of the operational requirements for the process control module. Their explanation was, *"Hey, we're not building a Cadillac!"* My reply was, "Sometimes you need that big old V8 engine to pull your heavy load up the hill." Over the next year, the development team added all of the capabilities that I described above by *hard-coding them into the message routing component*, one at a time, as they discovered the need for them. The expense was enormous, the message routing component grew to be 5 to 6 times the size that it should have been, and it arguably ran slower because all of the message options were being handled for each message on the normal timeline.

To add to the pain, they still had to implement the process control requirements that I had originally written in order to be able to control their large, complex system. Even worse, the company eventually decided that they wanted to "productize" these middleware components, but discovered that the cost of "modularizing" the capabilities was prohibitive since they were all tightly integrated into the message-routing component. They avoided the immediate cost of building a "Cadillac," they but they ended up paying the much higher lifecycle cost of building a tank that was too costly and cumbersome to reuse. This cost their company many millions of dollars in development costs and lost opportunities.

Another company I had dealings with came up with a proprietary scheme for handling encryption keys. Information could be encrypted in such a way that people with different security credentials could access only the portions of the information for which they had the proper credentials. This scheme cried out to be used in communications applications where you send out a single stream of information over a limited bandwidth network to multiple remote users, and they can decrypt only the parts that they are allowed to access, such as incremental increases in resolution of jpeg images or video streams. Unfortunately, the company had a vice president who used to be a big man at Microsoft in the area of desktop applications. You could ask him anything you wanted about Microsoft desktop applications and he could give you the answers. So naturally, this company chose to apply their technology to desktop applications. They convinced their investors to put up many millions of dollars to develop these applications, such as a capability for a word-processor to encrypt individual paragraphs separately, with the result that a user with limited access would see a redacted document. They could also create business forms with different fields separately encrypted so that different parts of a company could be restricted to only seeing the portions of the form for which they had access.

However, nobody bothered to ask the question, *"Why would you want to do this?"* Have you ever tried to read a redacted document? The only people who can really understand them fully are the ones who have full access. Deleting whole paragraphs here and there tends to obscure the context of the paragraphs that you can read. And to the extent that you can understand those paragraphs anyway, you can make inferences about the parts that have been cut out. As for using this technology to implement business forms, it requires that each piece of information be separately labeled so that you know how to decrypt. This can have a significant impact on the size and performance of the database as well as on the design of your database applications.

You've probably never heard of this company – I'm sure their investors wish they hadn't. Last time I checked, they were barely breathing.

## *Conclusion*

So how do you avoid disaster in the technology acquisition minefield? The same way that you avoid getting blown up in a regular minefield. You learn to recognize what the dangers look like, and you pay close attention to where you step. This usually involves using people with a deep understanding of all three aspects of product development and lifecycle cost to advise you.

Where do you find such people? How do you recognize them? Well, there aren't that many of them around. In the field of software development (my area of expertise), there are people who call themselves "software architects." These people claim to see the big picture. Then there are the programmers, who actually write the code. These people see the details. The really good software people are able to “zoom out” to see the big picture, and “zoom in” to understand the details of how the parts of the big picture are implemented. They should be able to do this seamlessly.

Really good software people understand baseline management and life cycle costs. They have *discipline*. Consider the following equation:

Life Cycle Cost <= Usability + Testability + Flexibility + Adaptability
+ Performance + Security + Reliability
+ Baseline Management + Risk Management

And, of course, engineering management manages the life cycle costs. Really good software people should be able to speak fluently on all of the elements of this equation. They should be able to give you the big picture, and they should also be able to zoom in on the details (if they can’t zoom in on the details, then you have to wonder whether or not their understanding of the big picture is really feasible).

You recognize such people by talking to them and listening to what they have to say. Trust your instincts. Does it sound like you are getting a snow job? If so, then you probably are. What kind of backgrounds do they have? Have they worked large projects? Have they worked small projects? You learn different things from each.

How many different types of business environments have they worked in? People with limited experience tend to have a limited view of the world, peppered with pre-suppositions about how things ought to work based on their narrow experience.

How many different types of applications have they worked on? The elements of the equation above represent a balancing act – a complex interplay of the different elements. You need people who understand this interplay and can manipulate it in such a way as to minimize the life cycle cost. You only gain this kind of understanding from broad experience.

The bottom line is, it’s your money, and bad technology has considerable leverage to waste it by the truckload. Don’t get trapped in the minefield. If you aren’t qualified to negotiate it yourself, find someone who is. The cost is minimal, and it has the potential to save you a fortune in the long run.